

AMENDMENTS TO THE SPECIFICATION:

On page 10, between lines 6 and 7 of the specification as filed, please insert the following paragraphs:

Another way to minimize cost and complexity is to exclude from the processor core functions that can be implemented with memory-mapped peripherals external to the core. For example, cache flushing may be performed using a small memory-mapped device controlled by a specialized software function. The cost and complexity of a data processor may also be minimized by implementing extremely simple exception behavior in the processor core.

A wide-issue processor pipeline, in contrast, executes bundles of operations in multiple stages. In a wide-issue processor, multiple concurrent operations are bundled into a single instruction and are issued and executed as a unit. In a clustered architecture, the machine resources are divided into clusters where each cluster consists of one or more register files each of which is associated with a subset of the execution units of the data processor.

A problem exists in that, to process these bundled instructions, the wide-issue processor pipeline consumes a large amount of power. For instance, a wide-issue processor will commonly execute "bundles" of operations in multiple stages, wherein

each stage in the pipeline is as wide as the executed word. Because it is generally not possible to completely populate a wide instruction with useful work (i.e., instructions), it is necessary to insert "dummy" instructions (i.e., non-operations) to fill all available slots. The problem arises in that these inserted "dummy" instructions consume power at each stage. Additionally, in normal operation, wide-issue processors require insertion of explicit non-operations to schedule correctly program execution (i.e., a feature of wide-issue processors over traditional sequential processors), and these non operations also consume power at each execution stage. As another example, power consumption problems can occur when repeated processor execution of small code sequences occurs as tight loops while unnecessary time is spent and power is expended in the cache.

Many data processors are not designed with a low/no power consumption mode, let alone functional units of the same, and, therefore, power consumption cannot be sufficiently reduced. Excessive power consumption by wide-issue data processors remains a continuing problem.

Please replace the paragraph on page 10 at lines 7–16 of the specification as filed with the following:

Therefore, there is a need in the art for improved data processors in which the cost and complexity of the processor core is minimized while maintaining the processor throughput. In particular, there is a need for improved systems and methods for executing conditional branch instructions in a data processor. More particularly, there is a need for systems and methods capable of addressing the problem of using remote branch conditions, while maintaining a local branch address computation, avoiding large amounts of global communication, and enabling a relatively good degree of scalability in the branch architecture. There is also a need in the art for improved data processors in which the cost and complexity of the processor core is minimized while maintaining the processor throughput. In particular, there is a need for improved systems and methods for reducing power consumption in a wide-issue data processor. More particularly, there is a need for systems and methods capable of addressing wasted power and time associated with unnecessary cache accesses.

Please replace the paragraph on page 15 at lines 9–10 of the specification as filed with the following:

FIGURES 2A and 2B illustrate ~~[[s]]~~ the exemplary data processor in greater detail according to ~~one~~ various embodiments of the present invention;

Please replace the paragraphs on page 16 at lines 1–6 of the specification as filed with the following:

FIGURE 7 illustrates a conceptual diagram of remote conditional branching control circuitry according to one embodiment of the present invention; ~~and~~

FIGURE 8 illustrates a flow diagram of an exemplary method of operating a data processor according to one embodiment of the present invention; and ~~[[.]]~~

FIGURE 9 illustrates a flow diagram of another exemplary method of operating a data processor according to one embodiment of the present invention.

Please replace the paragraph on page 17 at lines 2–9 of the specification as filed with the following:

FIGURES 1 through ~~8~~ 9, discussed below, and the various embodiments used to describe the principles of the present invention in this patent document are by way of illustration only and should not be construed in any way to limit the scope of the

invention. Those skilled in the art will understand that the principles of the present invention may be implemented in any suitably arranged data processor supporting a clustered architecture.

Please replace the paragraph on page 19 at lines 12–19 as follows:

FIGURES 2A and 2B are ~~is a~~ more detailed block diagrams of exemplary data processor 100 according to ~~one~~ various embodiments of the present invention. ~~Ø~~In the embodiment of FIGURE 2A, data processor 100 comprises instruction fetch cache and expansion unit (IFCEXU) 210, which contains instruction cache 215, and a plurality of clusters, including exemplary clusters 220-222. Exemplary clusters 220-222 are labeled Cluster 0, Cluster 1 and Cluster 2, respectively. Data processor 100 also comprises core memory controller 230 and interrupt and exception controller 240.

On page 21, between lines 18 and 19 of the specification as filed, please insert the following paragraphs:

As shown in FIGURE 2B, data processor 100 may also comprise power-down controller 250. Exemplary power-down controller 250 monitors the instruction cache 220 and the instruction execution pipeline of clusters 220-222 to identify power-down conditions associated with the same and, in response to an identified power-down condition, at least one of: (i) bypasses performance of at least a portion of subsequent processing stages associated with an executing instruction, (ii) powers down the instruction cache 220, or (iii) powers down the data processor 100. According to this embodiment, data processor 100 further comprises an instruction fetch buffer or instruction buffer (introduced hereafter), and power-down controller 250 operates to detect the presence of at least one of: (i) a non-operation in one of clusters 220-222, (ii) a tight-loop condition in the instruction fetch buffer, or (iii) an idle-loop condition.

In the embodiment illustrated in FIGURE 1B, power-down controller 250 is shown disposed within data processor 100. It will be appreciated by those skilled in the art that this particular configuration is shown by way of illustration only and should not be construed so as to limit the scope of the present invention in any way. In alternative embodiments of the present invention, all or a portion of power-down controller 250 may be externally associated with data processor 100.

On page 28, between lines 1 and 2 of the specification as filed, please insert the following paragraphs:

According to the embodiment illustrated in FIGURE 2B, power-down controller 250 monitors instruction cache 215 and instruction execution pipeline 400 to identify power-down conditions associated with the same. Three identifiable power-down conditions are (i) a non-operation in instruction execution pipeline 400, (ii) a tight-loop condition in instruction fetch buffer 305, or (iii) an idle-loop condition.

Power-down controller 250 detects a non-operation in instruction execution pipeline 400 in two ways. First, with respect to real non-operations (i.e., non-inserted NOPs), power-down controller 250 identifies the same while decoding is undertaken. Second, with respect to inserted non-operations, power-down controller 250 identifies the same at dispersion (i.e., at the time of insertion). This may advantageously be implemented in hardware.

Power-down controller 250 detects a tight-loop condition in instruction fetch buffer 305 at instruction decode when tight loops can be defined as those fitting within instruction fetch buffer 305 - those that fit within instruction fetch buffer 305

are recognized by the jump displacement and buffer sizing. This may advantageously be implemented in hardware.

Power-down controller 250 detects an idle-loop condition by determining whether the instructions in the tight loop in instruction fetch buffer 305 are non-operations (i.e., if all non-operations, then the tight loop may accurately be considered an idle loop). This may advantageously be implemented in hardware.

On page 28, between lined 8 and 9 of the specification as filed, please insert the following paragraph:

An important aspect of the present embodiment is that power-down controller 250 automatically powers down key circuitry in response to recognition of one or more power-down conditions as above-described.

On page 29, between lines 4 and 5 of the specification as filed, please insert the following paragraph:

An important aspect of the present embodiment is that power-down controller 250 automatically powers down key circuitry in response to recognition of one or more power-down conditions as above-described.

On page 30, between lines 4 and 5 of the specification as filed, please insert the following paragraph:

An important aspect of the present embodiment is that power-down controller 250 automatically powers down key circuitry in response to recognition of one or more power-down conditions as above-described.

Please replace the paragraph on page 36 at lines 12–17 of the specification as filed with the following:

To begin, data processor 100 enters fetch stage 402 first, generating an instruction fetch address (process step 805; FADDR) and then enters decode stage 403 second. During decode stage 403, instruction buffer 305 of FIGURE 3 receives instructions as 128-bit wide words from instruction cache 215 and the instructions are dispatched to a cluster 220-222 of FIGURES 2A and 2B (process step 810).

On page 39, between lines 6 and 7 of the specification as filed, please insert the following paragraphs:

FIGURE 9 illustrates a flow diagram (generally designated 900) of an exemplary method of operating data processor 100 to power down selected portions of data processor 100 according to one embodiment of the present invention. For purposes of illustration, concurrent reference is made to the exemplary embodiments of FIGURES 1 to 4.

To begin, data processor 100 executes instructions in clusters 220-222 (process step 905), wherein each cluster 220-222 comprises an instruction execution pipeline having seven processing stages, namely, address generation stage 401, fetch stage 402, decode stage 403, read stage 404, first execution (E1) stage 405, second execution (E2) stage 406, and write stage 407. Each of the seven processing stages 401-407 is capable of performing at least one of a plurality of execution steps associated with instructions being executed by clusters 220-222.

Power-down controller 250 monitors instruction cache 215 and each instruction execution pipeline 400, including instruction buffer 305 (process step 910). According to the present embodiment, during decode stage 403, instruction

buffer 305 receives instructions from instruction cache 215 and the instructions are dispatched to a cluster 220-222.

Power-down controller 250 operates to identify power-down conditions associated with instruction cache 215 and each instruction execution pipeline 400, namely (i) a non-operation in instruction execution pipeline 400, (ii) a tight-loop condition in instruction fetch buffer 305, or (iii) an idle-loop condition.

Power-down controller 250 monitors each instruction for the presence of non-operations (decision step 915). Exemplary power-down controller 250 detects a non-operation in instruction execution pipeline 400 in two ways. First, with respect to real non-operations (i.e., non-inserted NOPs), power-down controller 250 identifies the same while decoding is undertaken. Second, with respect to inserted non-operations, power-down controller 250 identifies the same at dispersion (i.e., at the time of insertion). This may advantageously be implemented in hardware. In the event that a non-operation is detected ("Y" branch of decision step 915), then power-down controller 250 bypasses performance of at least a portion of subsequent processing stages associated with the executing instruction having the non-operation (process step 920), thereby reducing power consumption in the subsequent processing stages as the executing instruction passes through the instruction execution pipeline (i.e., stages 404-407).

Power-down controller 250 monitors instruction fetch buffer 305 for the presence of tight loops (decision step 925). Exemplary power-down controller 250 detects a tight-loop condition in instruction execution pipeline 400 by monitoring instruction fetch buffer 305 at instruction decode when tight loops are defined, namely loops fitting within instruction fetch buffer 305 (i.e., recognized by the jump displacement and buffer sizing). This may advantageously be implemented in hardware. In the event that a tight-loop condition is detected ("Y" branch of decision step 925), then power-down controller 250 operates to power down instruction cache 210 (process step 930; e.g., responsive to identifying a tight-loop condition in instruction fetch buffer 305) pending termination of the tight loop.

Power-down controller 250 monitors each tight loop for the presence of idle loops (decision step 935). Exemplary power-down controller 250 detects an idle-loop condition in instruction execution pipeline 400, illustratively by monitoring instruction fetch buffer 305 to determine whether the instructions in the tight loop in instruction fetch buffer 305 are non-operations (i.e., if all non-operations, then the tight loop may accurately be considered an idle loop). This may advantageously be implemented in hardware. In the event that an idle-loop condition is detected ("Y" branch of decision step 935), then power-down controller 250 operates to power

down data processor 100 (process step 940), thereby stalling data processor 100
pending an interrupt.